الباب
الرابع

4

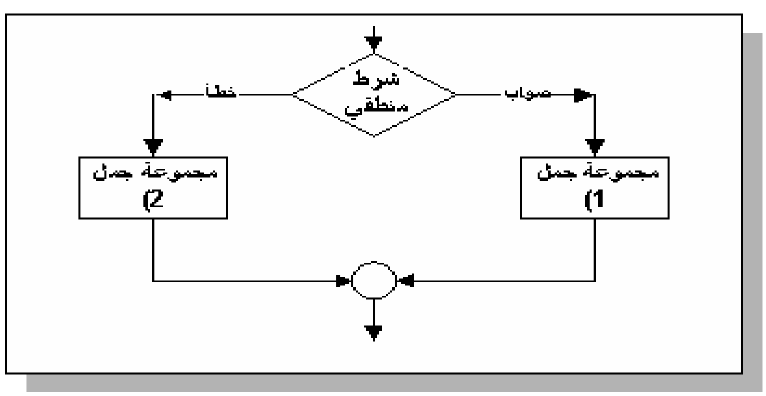
الجمل الشرطية

Conditional Statements

4.1 مقدمــة

تستخدم الجملة الشرطية في أي لغة لتحديد أكثر من مسار في أداء عمل ما. فعندما نقــرأ الجملة الشرطيــة : (إذا اجتهدت تنجح) نجـــد أن لــدينا اختيارين (مسارين): إما الاجتهاد الذي يؤدى إلى النجاح ، أو العكس .
وقد تسلك الجملة الشرطية أكثر من مسارين ، كأن نقول: (إذا كانــت درجــة الطالب 85 أو أكثر فتقديره ممتاز ، وإذا كانت من 75 إلى أقل من 85 فتقديره جيد جدا وإذا كانت من 65 إلى أقل من 75 فتقديره جيد ..... الخ) . مــن الواضح في هذا المثال أن هناك عدة مسارات نسلك أحدها بناء علــى درجــة الطالب لنعرف تقديره .
وذلك يعنى أن هناك نوعين من التفرع :

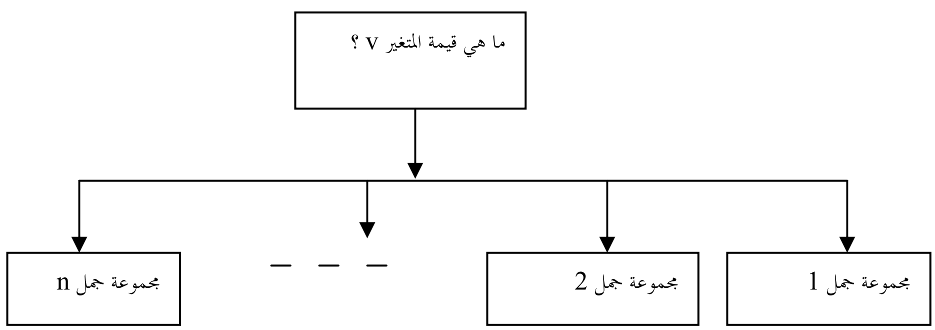
1 **<u>التفرع الثنـائي</u>** : حيث بناء على تحقيق شرط منطقي معين يتم أخــذ مسار واحد من مسارين على النحو التالي :



الشكل 4.1.1) تفرع ثنائي

في الشكل (4.1.1) يتم التوجيه إلى مجموعة الجمل (1) إذا تحقق الشرط المنطقي (True) أو يتم التوجيه إلى مجموعة (2) إذا لم يتحقق (False) .
لاحظ أن المقصود لمجموعة الجمل هو جملة واحدة أو أكثر قابلة للتنفيذ مثل تعيين قيم لمتغيرات أو قراءتها أو كتابتها ...

2 **التفرع المتعدد** : حيث يمكن أن يوجد أكثر من مسارين (اختيارين) ، يتم التوجيه إلى أحدها بناء على قيمة متغير واحد. أي أن المخطط الانسيابي لهذا التفرع يكون على النحو التالي:



الشكل (4.1.2) تفرع متعدد

في لغة سى تستخدم الجملة الشرطية (if - else) في حالة التفرع الثنائي ، وتستخدم جملة (switch - case) للتفرع المتعدد . سوف نتعرض في هذا الباب بصورة خاصة لهذا الموضوع ، ولكن قبل ذلك يجب أن ندرس العبارات المنطقية.

4.2 المؤثرات المنطقية logical operators

العبارة المنطقية هي العبارة القابلة للصواب true أو الخطــأ false، فمــثلا
العبارة : 5 أكبر من 4
هي عبارة منطقية وقيمتها الصواب true . والعبارة :

4 تساوى 3

هي أيضا منطقية وقيمتها false .

لاحظ هنا استــــخدام المقارنات مــــثل (أكبر من) و (يساوى) و (أصغر من) .... الخ في العبارات المنطقية ، لـــــذلك فقد خصص لها في لغة ســـى رموز خـــاصة تــسمى بالمؤثرات المنطقية logical operators كمـــا فـــي الجدول التالي :

المؤثر	المعنى
X > Y	X أكبر من Y
M < N	M اصغر من N
X = =Y	Y تساوي X
P != Q	Q لا تساوي P
Z >= W	Z أكبر من أو تساوي W
A <= B	A أصغر من أو تساوي B

الجدول (4.2.1)

مثال <u>(4.2.1)</u> **:** ماذا يطبع البرنامج التالي ؟

```
main()
{
	int x,y,z;
	x= (5>4)  ;
	y= (6<2);
	z= (8>=7);
	printf("\n %d %d %d ",x,y,z);
}
```

الشكل (4.2.1)

عند تنفيذ هذا البرنامج نحصل على الناتج التالي :

1 0 1

وهذا يعنى أن :

قيمة (4 < 5) هي 1

قيمة (2 > 6) هي 0

قيمة (7 =< 8) هي 1

حيث الصواب = 1 (true) و الخطأ = 0 (False)

ماذا لو لم نستخدم الأقواس في الجملة:

x = (5 > 4) ;

وكتبناها على الشكل:

x = 5 > 4 ;

هل تؤدى نفس الغرض ؟

الجواب: نعم ، لأن المؤثر > له الأسبقية على المؤثر = ، أي أن العـــملية (4 < 5) والتي قيمتها 1 تتم أولا ثم تتعين القيمة 1 إلى المتغير x .

مثال <u>(4.2.2)</u> : ماذا يطبع البرنامج التالي؟ :

```
main()
{
        int k,m,n;
        k=8>7>6;
        m=5>1+2;
        n=3*2>5;
        printf("\n %d %d %d ",k,m,n);
}
```

الشكل (4.2.2)

عند تنفيذ هذا البرنامج سيطبع الآتي :

0 1 1

ومعنى ذلك أن :

قيمة (6 > 7 > 8) هي 0

وقيمة (2 + 1 > 5) هي 1

وقيمة (5 > 2 * 3) هي 1

أي أن المؤثر > يتم تقييمه من اليسار إلى اليمين ، أي أن العبارة :

(8 >7)

تقيم أولا (وهى 1) ثم العبارة :

(1 > 6)

التي قيمتها 0 .

أما المؤثر + فيسبق المؤثر > ، لذلك يتم تقيـيم العبـارة (2 + 1) أي 3، ثـم المقارنة (3 > 5) وهى true أي 1 .

وكذلك فإن المؤثر * سبق المؤثر > بحيث تم تقييم 2 * 3 = 6 ثم المقارنة (6 > 5) وكانت النتيجة = 1 .

في الملحق تجد جدولا لجميع المؤثرات في لغة سى وأسبقياتها ، ويكفى هنا أن نلاحظ أن المؤثرات الحسابية لها الأسـبقية عـلى المـؤثرات المنطقيـة (العلائقية) .

4.3 التفرع الثنائي if – else

بالإمكان توجيه الحاسوب في البرنامج لسلوك أحد مسارين وذلك عن طريق جملة if. هذه الجملة تأخذ الشكل العام :

```
if  ( expression )
   block (1)
 else
   block (2)
```

حيث :

expression : هو عبارة منطقية قيمتها إما 0 أو 1

block(1) : مجموعة جمل يتم تنفيذها في حالة expression ! = 0
أي لا تساوى صفرا .

block(2) : مجموعة جمل يتم تنفيذها في حالة expression = = 0

لاحظ أن expression عادة يكون عبارة منطقية قيمتها 1 (في حالة الصواب) أو(في حالة الخطأ) . ولكن ليس بالضرورة أن يكون المتغير expression عبارة منطقية بل يمكن أن يكون عبارة حسابية .

مثال (4.3.1) : ماذا يطبع البرنامج التالي؟ :

```
main()
{       float x, y , z;
        printf("\n enter 2 numbers-->");
        scanf("\n%f,%f", &x, &y);
        if ( x>y )
                z=x;
        else
                z=y;
        printf("\n %f", z);
}
```

الشكل (4.3.1)

: 5 6

ب. في حالة إدخال العددين : 7.8 4.3

الإجابة :

في كلـــتـــا الحالتين سيطبع العدد الأكبر ، أي في الحالة الأولى سيطبع 6 وفى الحالة الثانية سيطبع 7.8 ، وذلك لأن z تـــساوى x عنـــدما (x > y) ، و z تساوى y عندما x < = y .

مثال (4.3.2) : اكتب برنامجا يقوم بقراءة الراتب الأساسي income

والعلاوة a ثم يحسب الراتب الإجمالي gross حيث :

gross = income + a

ويحسب الضريبة tax كالآتي :

- إذا كان الراتب الإجمالي أكبر من 500 تحسب عليه 20 % بالإضافة إلى ضريبة ثابتة invest قيمتها 25 ديناراً.
- وإلا (أي إذا كان 500 أو أقل) تحسب عليه ضريبة 15 % من الراتب الإجمالي بالإضافة إلى ضريبة ثابتة قيمتها 18 ديناراً.

ثم يحسب صافى الراتب net بخصم الضريبة من الراتب الإجمالي.

البرنامج المطلوب مبين بالشكل (4.3.2) .
فى هذا البرنامج نلاحظ أن الجملة التي تنفذ عند تحقق الشرط

gross > 500

هي:

tax = 0.2 * gross + 25 ;

وبالمثل ، فإن الجملة التي تنفذ عندما لا يتحقق الشرط

(gross > 500)

هي:

tax = .15*gross + 18;

```
main()
{    float income, a , tax, net, gross;
        printf("\n enter income-->");
        scanf("%f",&income);
        printf("\n enter allowance-->");
        scanf("%f",&a);
        gross = income +a ;
        if( gross > 500 )
                tax = 0.20*gross + 25;
        else
                tax = 0.15*gross + 18 ;
        net = gross - tax ;
        printf("\n\n gross=%.3f tax=%.3f
            net=%.3f",gross, tax, net);
}
```

الشكل (4.3.2)

4.4 التفرع المتعدد switch – case

تستخدم جملة switch - case في حالة وجود مسارات متعددة. وتأخذ جملة switch - case الشكل العام التالي :

```
switch  ( c)
   {        case  V1 : block 1  ;
```

```
        case  V2 : block 2  ;
         .....
        case Vn : block n  ;
       default : block d
}
```

حيث

V1 , V2 , ... Vn

هي القيم المحتملة للمتغير c ،

وحيث

block 1 , block 2 , ... block n

كل منها تمثل مجموعة من الجمل يتم تنفيذها كالآتي :

إذا كانت c = V1 يتم تنفيذ مجموعة الجمل block 1

إذا كانت c = V2 يتم تنفيذ مجموعة الجمل block 2

وهكذا ..

أما إذا لم يتحقق أي من الحالات المذكورة ، فيتم تنفيذ مجموعة الجمل:
block d

المحددة مع الحالة الافتراضية default . لاحظ أن مجموعة الجمل يجب أن تنتهي بالأمر break ، كما موضح بالمثال التالي :

مثال (4.4.1) : اكتب برنامجا لحساب الراتب salary من الدرجة scale على النحو التالي :

أ . في حالة الدرجة 1 ، الراتب = 200

ب . في حالة الدرجة 2 ، الراتب = 303

ج . في حالة الدرجة 3 ، الراتب = 456

علما بأنه لا يوجد إلا هذه الدرجات الثلاث ، وإذا تم إدخال عدد صحيح غير 1 أو 2 أو 3 يجب أن يصدر البرنامج تحذيرا بذلك .

يبين الشكل (4.4.1) البرنامج المطلوب، ومن أهم الملاحظات في هذا البرنامج استخدام جملة:

break ;

في نهاية كل مجموعة جمل ، وهى تعنى الخروج من جملة switch - case ، أي أنها كلمة مفتاحية keyword لها معنى خاص في مترجم لغة سى، وسوف نستخدمها في مواضع أخرى من هذا الكتاب ، مثل الخروج من دورة while ومن دورة for .....

```
main()
{      int scale;       float salary;
        printf("\n what is your scale-->");
        scanf("%d",&scale);
        switch(scale)
        {   case  1: salary=200;
                      break;
            case  2: salary = 303;
                      break;
            case  3: salary =456;
                      break;
           default : printf("\n ERROR!
                     Enter 1 or 2 or 3 only");
        }
        printf("\n your salary is %5.2f",salary);
}
```

الشكل (4.4.1)

لو ألغينا الأمر break مثلا من البرنامج السابق وكتبنا:

```
      switch ( scale )
    {     case  1  :  salary  =  200  ;
          case  2  :  salary  =  303  ;
        case  3  :  salary  =  456  ;
  default  :  printf ( " error ! ..... " ) ;
     }
```

سنجد أن الناتج هو :

your salary is 456.00

حتى لو أدخلنا قيمة 1 للمتغير scale . والسبب هنا واضح وهو أن الحاسوب لم يخرج من هذه الجملة الشرطية بعد تنفيذه للجملة :

salary = 200;

بل استمر لتنفيذ الجملة:

salary = 303;

ثم إلى الجملة التي تليها (salary = 456;) . وهذا خطأ شائع ويجب التنبه له .

نعود الآن – بعد هذه الملاحظة – إلى البرنامج بالشكل (4.4.1) حيث نلاحظ أن التنفيذ يتم على الصورة:

what is your scale → 1

حيث تم إدخال الرقم 1 للدرجة scale ، فيكون الناتج هو:

your salary is 200.00

وعند إدخال 2 أو 3 يعطى الراتب (salary) المقابل لهذه الدرجة . ولكن إذا أدخلت عددا غير 1 أو 2 أو 3 فإنه يرد بالرسالة التالية:

ERROR !
Enter 1 or 2 or 3 only

إذا أردنا استخدام if والاستغناء عن switch - case في هذا البرنامج ، فإن ذلك ممكن ولكنه غير مريح خاصة إذا تعددت الاختيارات ، ولا ننصح به .
ويبين الشكل (4.4.2) البرنامج مكتوباً باستخدام if بدلاً من switch-case ، ويتضح فيه أفضلية استعمال switch - case على if كلما تعددت الاختيارات.

4.5 المؤثر " أو " ||

تواجهنا أحياناً أكثر من حالة تتطلب نفس الإجراء. مثلاً قد نريد إعفاء الذين تقل أعمارهم عن 18 أو تزيد عن 65 سنة من دفع الضريبة. في هذه الحالة يكون الشرط المنطقي الذي يشمل الحالتين على النحو :

(س < 18 أو س > 65)

```
main()
{       int scale;
        printf("\n What is your scale?-->");
        scanf("%d",&scale);
        if(scale==1) printf("\n your salary is %5.2f",200.);
        if(scale==2) printf("\n your salary is %5.2f",303.);
        if(scale==3) printf("\n your salary is %5.2f",456.);
        if(scale>3)
```

```
    printf("\n ERROR! Enter 1 or 2 or 3 only");
}
```

الشكل (4.4.2)

حيث س تمثل العمر . وتكون الجملة الشرطية على النحو التالي :

إذا كانت (س < 18 أو س > 65) الضريبة = صفر

في لغة سى يستخدم المؤثر || (ويسمى مؤثر or) في مكان " أو " ، وبـــذلك تكون الجملة الشرطية المناظرة في لغة سى هي :

```
if(age < 18 || age > 65 ) tax = 0 ;
```

حيث استخدمنا المتغير age للعمر، و tax للضريبة.
إذا كنت من المبرمجين بلغة فورتران فإن المؤثر OR هو المـــألوف لـــديك . ويمكنك أيضاً استخدامه في لغة سى ولكن عليك أن تقوم بتعريفه فـــي البدايـــة على النحو :

```
#define OR ||
```

مثـال <u>(4.5.1)</u> : اكتب برنامجا يقوم بقراءة درجة الطالب grade ، بحيـــث يطبع الرسالة :

DATA ERROR !

إذا كانت الدرجة اكبر من 100 أوأقل من 0 ، وإلا ، إذا كانت الدرجة أكبر من أو تساوى 50 فيطبع كلمة (PASS) أي ناجح، أما إذا كانت الدرجة أقل من 50 فيطبع كلمة (FAIL)أي راسب.

```
main()
{   float grade;
        printf("\n Please enter grade(0 to 100)>");
        scanf("%f",&grade);
        if(grade<0 || grade>100)
                printf("\n DATA ERROR!!!!!!");
        else
                if(grade>=50)
                        printf("\n PASS");
                else
                        printf("\n FAIL");
}
```

الشكل (4.5.1) برنامج يستخدم المؤثر ||

لاحظ أن الغرض من الرسالة (DATA ERROR !) هو تنبيه المستخدم إلى إمكانية حدوث الخطأ في إدخال الدرجة، وهذا نوع من تحقيق البيانات data verification الذي يجب أن يتبعه المبرمج ليتأكد من عملية إدخال البيانات بطريقة صحيحة.

4.6 المؤثر " و " &&

يستخدم المؤثر && في العبارات المنطقية بمعنى (و) ، ويرمــز لــه عــادة بالرمز AND . وكما ذكرنا سابقاً يمكننا استخدام AND بدلاً من && عــن طريق التوجيه

```
#define AND &&
```

أما في لغة الرياضيات عندما نقول أن:

5 > س > 2

فذلك يعنى أن س أقل من 5 وأكبر من 2 . وبلغة سى تكون العبارة كالآتي :

```
x > 2 && x < 5
```

لاحظ أن الفرق الأساسي بين المؤثر " || " و المؤثر " && " هو أن:

```
1 || 0 = 0 || 1 = 1
```

بينما:

$$1 \&\& 0 = 0 \&\& 1 = 0$$

مثال (4.6.1) : اكتب برنامجاً لحساب الضريبة كالآتي :

أ . إذا كان الدخل 400 أو أقل يعفى من الضريبة .

ب . إذا كان الدخل أكبر من 400 أو أقل من 800 فعليه ضريبة 15 % من الدخل.

ج . إذا كان الدخل 800 أو أكبر فعليه ضريبة 20 % من الدخل.

```
main()
{
float income, tax;
printf("\n please enter income--->>");
scanf("%f",&income);
if(income <= 400 ) tax=0;
if(income > 400 && income < 800) tax=0.15*income;
if(income > 800) tax= 0.2*income;
printf("\n tax=%6.3f",tax);
}
```

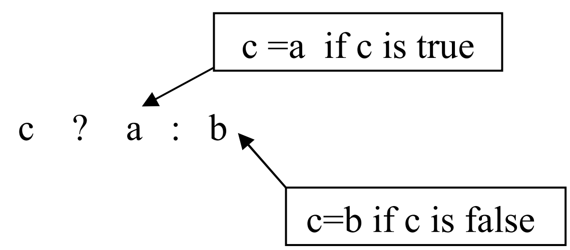
الشكل (4.6.1) برنامج يستخدم المؤثر &&

4.7 المؤثر الشرطي : ?

المؤثر ? يكون مصحوباً دائماً بعبارتين a و b تفصل بينهما الشارحة : (colon) على النحو التالي :

c ? a : b

حيث c متغير منطقي (قيمته إما true=1 أو false=0) بحيث إذا تحقق الشرط c تتعين قيمة a إلى c . و إذا لم يتحقق الشرط c (أي غير صحيح false) تتعين قيمة b إلى c . ويمكن التعبير عن هذا المعنى بالشكل التالي:



أي أن الجملة (c ? a : b) تعتبر اختصاراً للجملة الشرطية .

if (c) c=a else c=b;

مثال (4.7.1) : ماذا يطبع البرنامج التالي؟:

```
main()
```

```
{   int x,c;
        printf("\n enter a number-->");
        scanf("%d",&x);
        c=(x>=50) ? 1 : 2 ;
        printf("\n %d", c);
}
```

الشكل (4.7.1) برنامج يستخدم المؤثر ?

هذا البرنامج يقوم باستقبال قيمة x ويقارنها مع العدد 50، فإذا كانت أكبر من أو تساوي 50 يعين القيمة 1 (أي القيمة التي على يسار الشارحة :) للمتغير c، وإذا كانت 50 أو أقل يعين القيمة 2 (أي التي على يمين الشارحة :) للمتغير c .

طبعاً بإمكاننا كتابة الجملة :

```
if (x >= 50 )
        c  = 1  ;
  else
        c  =  2  ;
```

بدلاً من الجملة:

```
c  =  ( x >= 50 )  ?  1 : 2  ;
```

ولكن الغرض من هذا المثال هو توضيح استخدام المؤثر ? في لغة سى .
ولا يفوتنا هنا أن نلاحظ خاصية من خصائص لغة سى وهى إمكانية كتابة جملة داخل جملة . فمثلاً بدلاً من أن نكتب جملتين كالآتي :

```
c = ( x >= 50 ) ;
   c ? 1 : 2    ;
```

نكتبهما في جملة واحدة هي :

```
c = ( x >= 50 ) ? 1 : 2 ;
```

4.8 تمارين

1. اكتب برنامجاً يقوم بقراءة عدد صحيح، ويطبع كلمة (BIG) إذا كان العدد أكبر من 1000، وكلمة (SMALL) إذا كان العدد 1000 أو أقل.

2. استخدم :
   ا. جملة switch - case
   ب. جملة if - else

لطباعة كلمة (positive) إذا كان العدد المدخل x > 0، وكلمة negative إذا كان العدد x < 0 ، وكلمة (zero) إذا كان العدد x = 0 .

3. اكتب برنامجاً يقوم بقراءة :

ا . الحالة الاجتماعية (متزوج أو غير متزوج) حيث:
m = 1 تعنى متزوج ، m = 0 غير متزوج .
ب . الدخل
ويحسب البرنامج الضريبة tax على النحو التالي:

- الدخل < 500 ومتزوج
  الضريبة = 10 % من الدخل
- الدخل > 500 ومتزوج
  الضريبة = 15 % من الدخل
- الدخل < 500 وغير متزوج
  الضريبة = 20 % من الدخل
- الدخل 500 وغير متزوج
  الضريبة = 25 % من الدخل

4. أعد كتابة التمرين (3) مع الأخذ في الاعتبار عامل السن age ، وذلك بإعفاء المسنين الذين تزيد أعمارهم عن 60 سنة من 100 دينار من قيمة الضريبة . وإذا أصبحت الضريبة قيمة سالبة تحول إلى صفر. استخدم المؤثر ? في هذا البرنامج.

5. اكتب برنامجا لحساب قيمة f التي تعتمد على قيمة x على النحو التالي:
   - إذا كانت x أكبر من 1 فإن قيمة f تساوي 2.
   - إذا كانت x تساوي 1 أو أقل فإن f تساوي مربع x.

6. اكتب برنامجاً يقوم بحساب العمر بالسنوات والأشهر وذلك بقراءة تاريخ الميلاد (الشهر و السنة) وتاريخ اليوم (الشهر والسنة) وطرح التاريخين.

7. أعد كتابة البرنامج في تمرين (6) وذلك بحساب الأيام إضافة إلى الأشهر والسنوات. افترض أن الشهر = 30 يوماً.

8. اكتب جزءا من برنامج للتحقق من إدخال التاريخ بصورة صحيحة، بحيث يحقق الآتي :

   اليوم لا يزيد عن 31

الشهر لا يزيد عن 12
السنة لا تزيد عن 3000 (مثلاً)
ثم أضف هذا الجزء إلى البرنامج المطلوب في تمرين (7) .

9 . اكتب برنامجاً يقوم بحساب الفترة الزمنية بالساعات والدقائق والثواني ما بين الوقت h1 : m1 : s1 و h2 : m2 : s2 حيث :

h1 , h2 الوقت بالساعات
m1 , m2 الوقت بالدقائق
s1 , s2 الوقت بالثواني

مثال : الفترة الزمنية من 40 : 25 : 2 إلى 15 : 10 : 5 هي :

ثواني	دقائق	ساعات
15	10	5
40	25	2
35	44	2

ملاحظة : استخدم جملة if - else

10. اكتب برنامجاً يقوم بإيجاد مجموع أعلى درجتين تحصل عليه طالب من بين 3 امتحانات. مثلاً إذا تحصل على الدرجات : 30 , 50 , 70 فإن المجموع المطلوب هو 50 + 70 = 120 .

11. اكتب معاني المصطلحات التالية:

branching
logical operator
relational operator
conditional operator-
data verification
logical expression

الباب الخامس

5

الحلقات

Loops

5.1 مقدمة

ندخل من هذا الباب إلى مجال من مجالات الحاسوب الهامة، والاستفادة من قدرته على تكرار العمل المطلوب بسرعة هائلة دون كلل أو ملل .

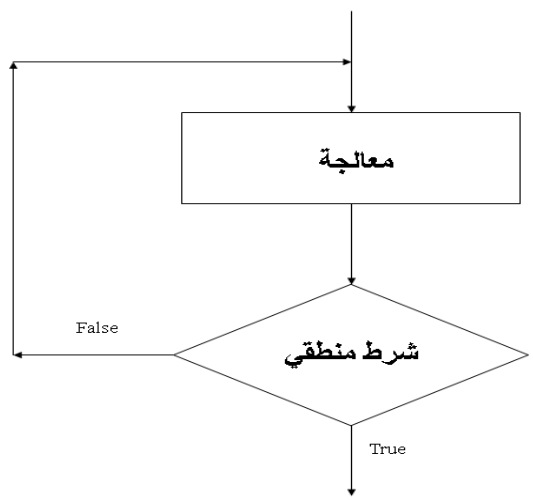
البرامج التي تم إعدادها في الأبواب السابقة لم يكن بها تكرار ، أو ما يعــرف بالـــدوران في حلقة looping ، بل كانت ذات اتجاه من فوق – إلى – أســـفل top-down مع احتمال التفرع branching إلى أكثر من مسار.

الآن نريد من الحاسوب أن يكرر عملاً معيناً إلى أن يتحقق شرط منطقي .

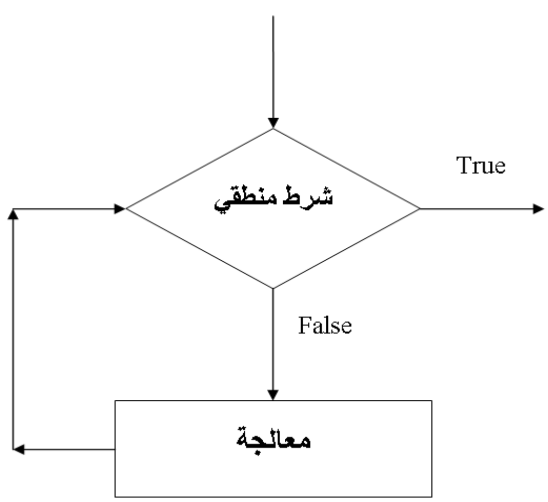
والشرط المنطقي ضروري في العمليات التكرارية ، فهو الوسيلة التي نستخدمها للتحكم في عدد الدورات. فمثلاً إذا طلب منك جمع الأعداد من 1 إلى 100 فإنك تبدأ من 1 ثم 2 ثم 3 ... وهكذا إلى أن تصل إلـــى 100 ، عنـــــدها تتوقـــف لأنـــك تسأل نفسك أثناء العمل (هل وصلت إلى 100 ؟) وتكون الإجابة (لا) إلى أن تصل العدد المطلوب.

عند إعداد خريطة انسيابية للحلقة ، قد يكون التحقق من الشرط المنطقـــي فـــي نهاية الحلقة كما في الشكل (5.1.1) . وقد يكون في بداية الحلقـــة كمـــا فـــي الشكل(5.1.2) .

الفرق الأساسي بين الـــشكلين (5.1.1) و (5.1.2) هـــو أن الـــشكل الأول يتطلب إجراء المعالجة ولو مرة واحدة على الأقل لأن اختبار الشرط المنطقـــي يتم بعد الـــــمعالجة ، أما الشكل الثاني فيسمح بعدم إجراء أي معالجة إذا لـــم يتحقق الشرط المنطقي إطلاقا .



الشكل(5.1.1) الشرط المنطقي في نهاية الحلقة.



الشكل (5.1.2) الشرط المنطقي في بداية الحلقة

5.2 دورة (طالما) while

تأخذ هذه الدورة الشكل العام التالي:

```
while ( condition )
    {
        statements;
    }
```

حيث تطلب تنفيذ الجمل (statements) المحصورة ما بين القوسين { } طالما تحقق الشرط المنطقي condition (يسمى شرط استمرار). أي أن أول عمل يقوم به الحاسوب عند تنفيذه لهذه الحلقة هو اختبار الشرط المنطقي، بحيث إذا كان صائباً true سينفذ الجمل المحصورة ، وإذا كان الشرط خاطئاً false يخرج من الحلقة.

> **<u>ملاحظة</u>** : يجب أخذ الحذر والتأكد من أن الشرط المنطقي لن يتحقق بعد عدد محدود من الدورات ، وإلا فإن الحاسوب سيدخل في حلقة لانهائية infinite loop لا يخرج منها إلا بإعادة التشغيل من جديد مما قد يسبب في تلف البرنامج إذا لم يحفظ.

<u>مثال</u> : اكتب برنامجاً لإيجاد مجموع الأعداد

$$1 + 2 + 3 + \ldots\ldots\ldots\ldots + 100$$

مستخدماً دورة (طالما).

يجب أولاً تحديد الشرط المنطقي لهذه الدورة . من الواضح هنا أن التوقف والخروج من الدورة يجب أن يتم عندما يتحقق شرط التوقف وهو

$$k > 100$$

حيث k متغير صحيح يرمز للأعداد من 1 إلى 100 . أي أن

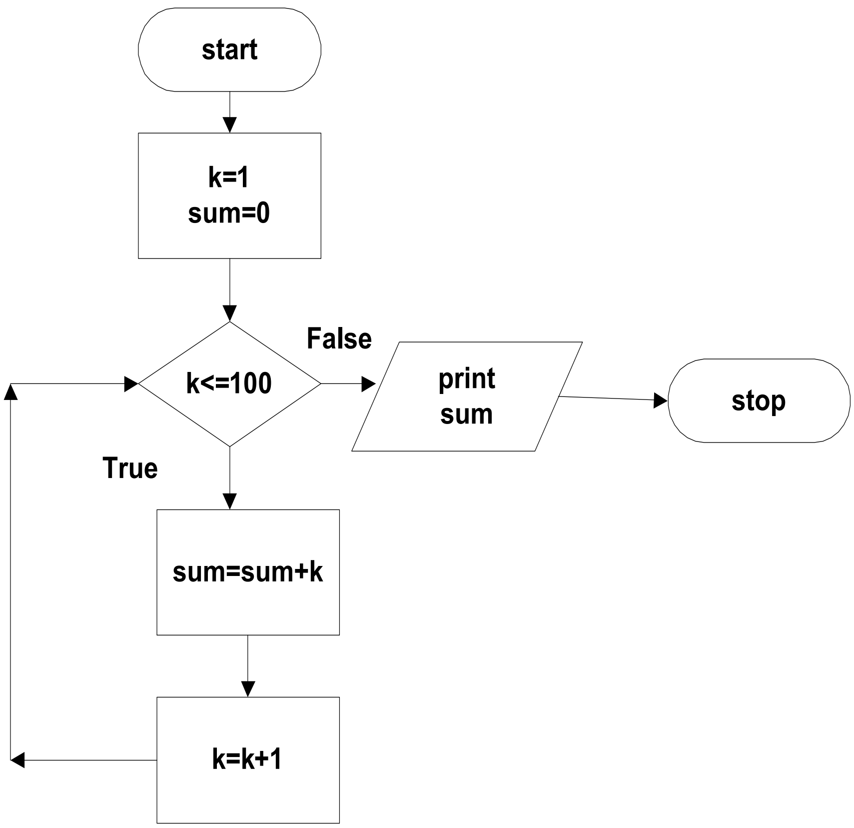
$$k = 1$$

هي نقطة البداية، ثم نضيف 1 إلى k ، ونضيف k إلى المجموع المطلوب وتستمر هذه العملية طالما تحقق شرط الاستمرار :

$$k <= 100$$

لاحظ أن شرط الاستمرار – وهو المستخدم في دورة (طالما) – يكون دائماً عكس شرط التوقف.

لـــذلك فإن الشكل الانـــسيابي لهذه المســـألة يكون علـــى النحـــو المبـــين بالشـــكل 5.2.2:



الشكل (5.2.2) مخطط انسيابي لحساب مجموع الأعداد من 1 إلى 100.

البرنامج المقابل لهذه الخريطة الانسيابية مبين في الشكل (5.2.3) .

```
#define N 100
main()
{
        float  sum=0;
        int k=1 ;
        while(k<=N);
        {
                sum = sum + k;
                k = k + 1;
        }
        printf ("\n sum=%d",sum);
}
```

الشكل (5.2.3) برنامج لجمع الأعداد من 1 إلى 100

عند تنفيذ هذا البرنامج سنجد أن الناتج هو :

sum = 5050

كما هو متوقع من الصيغة:

sum = n(n+1)/2 = 100(101)/2=5050

كان طبعاً من الممكن استخدام هذه الصيغة في البرنامج والاستغناء عن الحلقة، ولكن الغرض هنا هو توضيح استعمال دورة (طالما) . والمثال التالي يعتبر أكثر واقعية ، فهو يتطلب إجراء دورة إما باستخدام (طالما) أو غيرها .

مثال : اكتب برنامجاً لإيجاد أكبر قيمة من بين مجموعة من القيم عددها n .

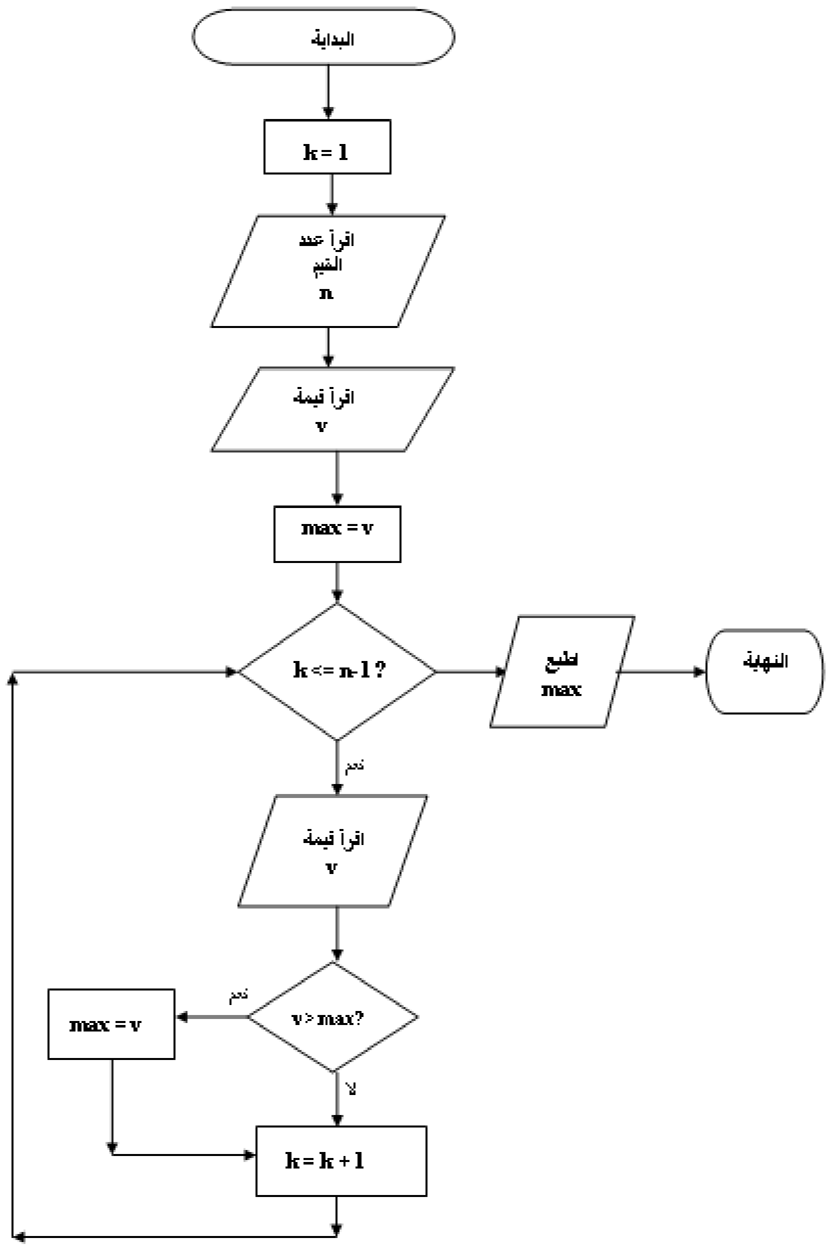
لاحظ أن المعطيات هنا هي عدد القيم n والقيم نفسها. لإيجاد أكبر قيمة نتبع المخطط الانسيابي كما في الشكل (5.2.4). في هذا الخريطة الانسيابية نجد دورة مناسبة لاستخدام أسلوب (طالما)، ولكن قبل الدخول الدورة علينا أن نعطى قيمة للمتغير max (الذي سيحمل في النهاية أكبر قيمة من بين القيم المدخلة) . بالإمكان تحديد القيمة الابتدائية بأنها صفر لأنها ستتغير عند المقارنة بالقيم الأخرى ، ولكن قد يحدث أن تكون كل القيم سالبة والصفر ليس من بينها ، مما يجعل أكبر قيمة تساوى صفراً وهذا غير صحيح.

لذلك فمن الأفضل أن تعطى القيمة الأولى من بين القيم المدخلة للمتغير max ثم يبدأ التبديل من القيمة الثانية فما بعد.

لاحظ أن شرط الاستمرار المنطقي في الحلقة (في المخطط الانسيابي والبرنامج) هو :

$$k <= n - 1$$

ذلك لأن عدد القيم الباقية بعد القراءة الأولى هو n - 1 وليس n .



شكل(5.2.4) مخطط انسيابي لإيجاد أكبر قيمة

لاحظ أن شرط الاستمرار المنطقي في الحلقة (فـــي المخطـــط الانـــسيابي والبرنـامج) هو :

$$k <= n - 1$$

ذلك لأن عدد القيم الباقية بعد القراءة الأولى هو n - 1 وليس n .

```
/*------------------PROGRAM EX524.C-----------*/
main()
{
        int k=1 ,n;
        float val, max;
        printf("\n Enter number of values→");
        scanf("%d",&n);
        printf("\n Enter a value-->");
        scanf("%f",&val);
        max= val;
        while( k<= n-1 )
        {    printf("\n Enter a value-->");
                scanf("%f",&val);
                if( val > max) max=val;
                k = k+1;
        }
        printf("\n\n Maximum value is %f",max);
}
```

الشكل (5.2.4) برنامج إيجاد أعلى قيمة .

مثـال : اكتب برنامجا لحساب متوسط درجات 20 طالبـــا فـــي مقـــرر دراسي واحد .

لاحظ أولاً أن:

المتوسط = مجموع الدرجات مقسوما على عدد الدرجات.

أو بالرموز:

average = sum / n

حيث average يرمز للمتوسط و sum للمجموع. أما الدرجة فنرمز لها بالرمز grade .

هذا البرنامج يشبه إلى حد ما برنامج إيجاد مجموع الأعداد من 1 إلى 100 ، ولكن الفرق يكمن في ضرورة قراءة الدرجات في هذا البرنامج ، وجمع كل درجة grade للمجموع sum مباشرة بعـــد قراءتهـــا ، ولا تـــنس أن تضيف 1 إلى العداد k في كل دورة.

لاحظ في هذا البرنامج أننا استخدمنا الثابت N لعدد القيم وهـــو 20 حتـــى يسهل استبداله برقم أخر إذا لزم الأمر.

ماذا لو نريد التحقق من الشرط المنطقي في نهاية دورة (طالما) ولـــيس في أولها ؟

نستطيع عمل ذلك بأن نجعل القيمة التي تتعين للشرط المنطقي ثابتة على 1 ، أي أن الشرط يكون دائما صائباً true ، ثم نستخدم الأمر break للخروج من الدورة مع جملة if .

```
#define N 20
main()
{
        float grade, sum=0, average;
        int k=1 ;
        while(k<=N)
        {
                printf("\n Please enter a grade-->");
                scanf("%f", &grade);
                sum = sum + grade;
                k = k + 1;
        }
        average = sum /N;
        printf ("\n average=%f",average);
}
```

الشكل (5.2.5) برنامج حساب المتوسط

فمثلاً البرنامج بالشكل (5.2.6) يضع شرط الاستمرار في نهاية الدورة أي بعد قراءة الدرجة وإضافتها للمجموع وإضافة 1 إلى العداد k . لاحظ أولاً أن هذا العداد يبدأ من k = 0 وليس من k = 1 لأن اختبار الشرط :

(k < = N)

```
#define N 20
main()
{       float grade, sum=0, average;      int k=0 ;
        while( 1)
        {
                printf("\n Please enter a grade-->");
                scanf("%f", &grade);
                sum = sum + grade;
                k = k + 1;
                if(k>=N)  break;
        }
        average = sum /N;
        printf ("\n average=%f",average);
}
```

الشكل (5.2.6) دورة (طالما) بالأمر break

يأتي بعد إضافة 1 للعداد . فمثلاً إذا كان عدد القيم هو 1 نكون قد انتهينا من القيم بمجرد إضافة 1 إلى 0 .

الملاحظة الأخرى هي أن الدورة :

```
while (1)
{ ......
      ......
}
```

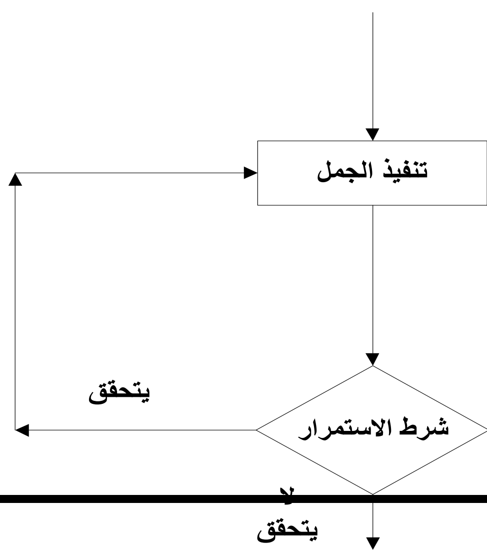
تعتبر دورة لانهائية infinite loop إذا لم نضع فيها جملة التوقف break .

بالإمكان أيضاً استخدام جملة go to للخروج من الحلقة ولكن هذه الجملة غير مفضلة لدى العاملين في مجال البرمجة ، ولا ننصح بها لأن استعمالها قد يسبب غموضاً وصعوبة في تتبع البرنامج .

والأصح أن نستخدم تركيبة أخرى تسمى do-while (أنجز – طالما) التي تتطلب وضع (شرط الاستمرار) في نهاية الدورة .

5.3 دورة (أنجز – طالما) do - while

تتطلب هذه الدورة تنفيذ الجمل قبل اختبار شرط الاستمرار كما في الشكل التالي:



ومعنى ذلك أن الجمل يجب أن تنفذ مرة واحدة على الأقل ، بخلاف دورة while حيث نجد أن الجمل قد لا يتم تنفيذها إطلاقا.

والشكل العام لهذه الدورة يكون على الصورة :

```
do
{ statements }
while ( condition ) ;
```

حيث :

statements : جمل الدورة
condition : شرط الاستمرار

وكالعادة يجب أن نلاحظ عدم ضرورة القوسين { } في حالة وجود جملة واحدة في الدورة .

<u>مثال</u> : نعيد الآن برنامج حساب المتوسط باستخدام do - while كما مبين بالشكل (5.3.2) .

لاحظ في هذا البرنـــامج أن القــيمة الابتدائية للعداد k هي 0 وليس،1 ذلك لأن زيادة العداد تأتى قبل اختبار شرط الاستمرار وليس بعده كما في دورة (طالما).

```
#define N 20
main()
{       float grade, sum=0, average;
        int k=0 ;
        do
        {
                printf("\n Please enter a grade-->");
                scanf("%f", &grade);
                sum = sum + grade;
                k = k + 1;
        }
        while(k<=N);
        average = sum /N;
        printf ("\n average=%f",average);
}
```

الشكل (5.3.2) برنامج حساب المتوسط باستخدام do - while

5.4 دورة for

تتميز هذه الدورة عن دورة طالما بإمكانية اشتمالها على الحالة الابتدائية ، وشرط الاستمرار ، وزيادة أو نقصان عداد الدورة في آن واحد، وذلك على النحو التالي:

```
for ( s1 ; condition ; s2 )
    { statements }
```

حيث:

s1 ترمز لجملة تعيين القيمة الابتدائية لعداد الدورة .

condition ترمز لشرط الاستمرار continuation condition .

s2 ترمز لجملة زيادة (أو نقصان) عداد الدورة .

لاحظ أنه ليس من الضروري تحديد هذه العبارات الثلاث جميعاً، بل يمكن أن يترك أي منها شاغرة ولكن يجب الإبقاء على الفاصلتين المنقوطتين.

<u>أمثلة</u> :

1 . الجملة

```
for ( sum = 0 , i = 1 ; i < = 10 ; i = i + 1 )
          sum = sum + i ;
```

تقوم بجمع الأعداد من 1 إلى 10 في المتغير sum.توجد هنا جملتان ابتدائيتان هما sum=0 و i=1 . أما شرط الاستمرار فهو i<=10 ، وجملة الزيادة هي i=i+1.

2 . الجملة

```
for ( m= 100 ; m > = 1 ; m = m - 1 )
printf (" \ n %d " , m )  ;
```

تقوم بطباعة الأعداد من 100 إلى 1 تنازلياً .

3 . الجملة

```
for ( k= 1 ;  ; k = k + 2 )
  {   printf ("\ n %d " , k )  ;
      if ( k > 50 )  break  ;
    }
```

تؤدى إلى طباعة الأعداد الفردية من 1 إلى 51 تصاعدياً .

لاحظ هنا أننا وضعنا (شرط التوقف) داخل الدورة بدلا من كتابة (شرط الاستمرار) الذي تركنا مكانه شاغرا بين الجملة (k = 1) و الجملة (k = k+ 2)

من المهم أن نلاحظ هنا أن دورة for تتبع الخطوات التالية :

1 . تعيين القيمة الابتدائية.

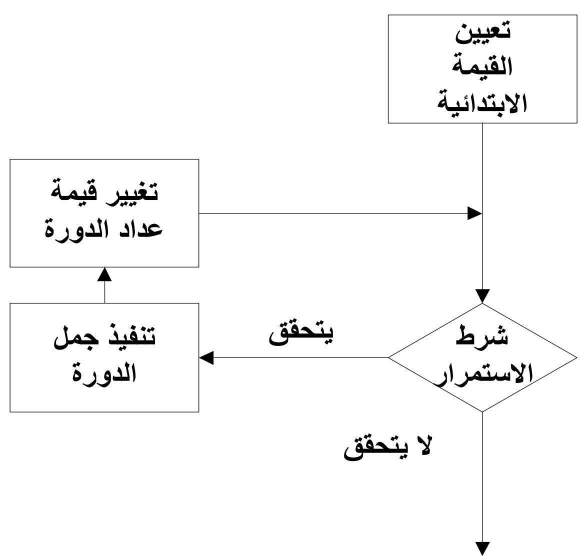
2 . اختبار شرط الاستمرار.

3 . تنفيذ جمل الدورة.

4 . تغير عداد الدورة.

5 . الرجوع إلى الخطوة (2) .

أي أن المخطط الانسيابي لهذه الدورة يكون كالآتي :



الشكل(5.4.1) المخطط الانسيابي لدورة for

كما يتضح من الأمثلة السابقة فإن دورة for تتميز عن دورة while بوضع العبارات الثلاث اللازمة لكل دورة (القيمة الابتدائية + شرط الاستمرار + تغير العداد) أمام كلمة for ، تبسيطاً للدورة ، التي تصبح متكونة أساساً من جمل الدورة فقط.

لاحظ أن دورة for تغني عن دورة while لأنه يمكننا استعمال دورة for على النحو التالي:

```
for ( ; condition ; )
```

وهي تعادل الجملة:

```
while (condition)
```

مثال : اكتب برنامجاً باستخدام دورة for لحساب أعلى درجة تحصل عليها طالب من بين 20 طالباً .

من الواضح في هذا المثال أن المدخلات هي 20 درجة ، وأن المطلوب إيجاد أعلى درجة، ونسميها max، كما نسمى الدرجات نفسها grade .
و بما أن الدرجات تكون في العادة كلها موجبة يمكننا أن نبدأ بالحالة:

max = 0

ثم تغير هذه القيمة – كما في الشكل (5.4.2) – كلما وجدنا درجة grade أكبر من max وتبديلها بهذه الدرجة .

```
#define NofG 20
main()
{
        float grade, max;
        int count;
        for(count=1, max=0 ; count <= NofG; count = count +1
)
        {
                printf("\n Please enter a grade-->");
                scanf("%f",&grade);
                if( grade>max ) max=grade;
        }
        printf("\n maximum=%f",max);
}
```

الشكل (5.4.2) دورة for لحساب أعلى درجة

مثـال : اكتب برنامجاً لحساب مضروب العدد n (ويرمز له عادة بالرمز n!) حيث

$$n! = n*(n-1)*(n-2)*\ldots*3*2*1$$

هذا النوع من المسائل مناسب لاستخدام دورة for . هنا نجد أن المطلوب هو حساب المضروب (ونسميه في البرنامج fact اختصاراً ل factorial) ويمكن استخدام الجملة:

```
fact = fact * k ;
```

على أن يبدأ من القيمة 1 (وليس من الصفر كما هو الحال في عملية الجمع). أما العداد k فيبدأ من الواحد ثم يزداد بمقدار واحد في كل دورة حتى يصل إلى n بعد n من الدورات.

```
main()
{
        int k , n , fact;
        printf("\n Please enter n-->");
        scanf("%d",&n);
        for( k=1 , fact=1 ; k<=n ; k=k+1)
                fact = fact*k;
        printf("\n factorial of %d is %d " , n,fact);
}
```

الشكل (5.4.3) برنامج حساب مضروب n

<u>ملاحظة</u> : عند تنفيذ هذا البرنامج يجب إدخال $n \leq 10$. لماذا ؟

<u>مثال</u> : اكتب برنامجاً لحساب المجموع :

sum = 1 + 1/2! + 1/3! + 1/4! + ... + 1/n!

قد يبدو لأول وهلة أن هذا البرنامج يتطلب حساب المضروب n من المرات ، ولكن ذلك يعتبر طريقة غير اقتصادية من حيث وقت الحاسوب.
و لتجنب ذلك، من الأفضل كتابة sum على الصورة التالية :

$$sum = T_1 + T_2 + T_3 + ... T_n$$

حيث :

$$T_k = 1/ k!$$

وبالتالي يمكن إثبات أن:

$$T_{k+1} = T_k / (k+1)$$

من الأفضل استخدام هذه العلاقة في حساب T_k إذ أنها أكثر توفيراً لوقت الحاسوب . في البرنامج نكتب هذه العلاقة على الصورة :

t = t / (k+1) ;

نبدأ أولا بالقيمة T = 1 و sum = 0 ، ثم نقوم بعملية الجمع ابتداء من

k = 1

إلى

k = n

على النحو التالي :

```
main()
{
   int k,n ;
   float t, sum;
   printf("\n Please enter n-->");
```

```
   scanf("%d",&n);
   for( k=1 , t=1 , sum=0;  k<= n;   k= k+1)
   {
            sum = sum +t;
             t = t /(k+1);
   }
   printf("\n sum =%f",sum);
}
```

الشكل (5.4.4) برنامج حساب المتسلسلة

لاحظ عند تنفيذ هذا البرنامج وإدخال قيمة 10 للمتغير n نحصل على الناتج :

sum = 1.718282

5.5 الحلقات اللانهائية infinite loops

ماذا يحدث إذا كتبنا جزءا من برنامج كالآتي ؟

```
        for  (  ;  ;  )
              printf (" Good Morning.  " )  ;
```

ما يحدث هو أن العبارة

Good Morning.

ستظهر على الشاشة ما لا نهاية من المرات، والسبب هو أننا لم نضع شرطا للتوقف في هذه الحلقة .

لذلك نسمى الحلقة :

```
for (  ;  ;  )
  { ....... }
```

بالحلقة اللانهائية infinite loop .

وحتى تكون الحلقة نهائية يجب وضع شرط للتوقف داخل الدورة مثل :

```
if ( k > 100 )  break   ;
```

أو :

```
if ( answer > 10 )  break  ;
```

مثال : ماذا يحدث عند تنفيذ البرنامج التالي ؟

```
main()
{
   char ans;
   for( ; ; )
   {
           printf("\n Press any key to
                         continue.");
           printf("\n Enter y to stop-->");
           scanf("%c",&ans);
           if(ans=='y')break;
```

```
   }
}
```

الشكل (5.5.1) الحلقة اللانهائية

تستخدم الحلقة اللانهائية عادة في التحقق من البيانات المدخلة، فمثلاً إذا كان المطلوب إدخال درجة طالب في مقرر دراسي، وكان نطاق الدرجة لا يقل عن الصفر ولا يزيد عن 100، فيمكننا إجراء الجمل التالية لتجنب الأخطاء في الإدخال :

```
for ( ; ; )
      { printf ( " \n enter grade →-> " ) ;
        scanf ( " %d " , &grade ) ;
        if ( grade > = 0 && grade < = 100 )
              break ;
      }
```

بهذه الطريقة لن يقبل الحاسوب إدخال درجة غير سليمة مثل 150 أو درجة سالبة. ولكن هذه الطريقة لا زالت تحتاج إلى تحسين، فالمستخدم لن يفهم لماذا لم يقبل الحاسوب الدرجة ولا يتحرك إلى غيرها. لذلك يكون من الأفضل كتابة شرط التوقف على النحو التالي:

```
if ( grade > = 0 && grade < = 100 )
      break ;
```

```
else
printf ( " \ n ERROR! data  out of range . " ) ;
```

حيث تظهر في حالة الخطأ الرسالة:

ERROR! data out of range

وتعنى أن هناك خطأ في البيانات خارج النطاق المطلوب.

5.6 مؤثرات التغير increment operators

تستخدم المؤثرات التالية :

+=   -=   =   /=   ++   --

بصورة خاصة في لغة سى كنوع من الاختصار في كتابة الجمل الحسابية التي تحدث تغييراً increment في قيمة متغير.

فمثلاً بدلا من كتابة الجملة:

```
k = k + 1   ;
```

(وهى تعنى أضف 1 إلى k) نكتبها بصورة مختصرة بالشكل :

```
k  += 1  ;
```

أي أن المؤثر += يقوم بإضافة قيمة الطرف الأيمن إلى المتغيـــر فـــى الطرف الأيسر.

وبطريقة أخرى ، يمكن كتابة الجملة المختصرة :

```
k++ ;
```

وهى تعنى أيضاً (أضف 1 إلى k)

بنفس الطريقة، فإن :

```
a -= b;
```

تعنى اطرح b من a، والجملة :

```
a *= b;
```

تعنى اضرب b في a وعيّن الناتج إلى a، والجملة :

```
a /= b;
```

تعنى اقسم a على b وعين الناتج إلى a، أما:

```
a-- ;
```

فتعنى اطرح 1 من a.

<u>**مثال**</u> : ماذا يطبع البرنامج التالي؟

```
main()
{       int a=1, b=2 , c=3 , d=4;
        a += b;
        b *= c;
        c -= a-2;
        d += b/c;
        printf("\n %d  %d  %d  %d", a,b,c,d);
}
```

الشكل (5.6.1) مؤثرات التغيير

الجملة الأولى

$$a += b ;$$

تعنى

$$a = a + b$$
$$= 1 + 2 = 3$$

والجملة

$$b *= c ;$$

تعني

$$b = b * c = 2 * 3 = 6$$

والجملة

$$c -= a - 2 ;$$

تعنى

$$c = c - (a - 2)$$
$$= 3 - 1 = 2$$

وأخيراً الجملة

$$d += b / c$$

تتطلب إجراء العملية

$$b / c = 6 /2 = 3$$

ثم إجراء العملية

$$d = d + 3 = 7$$

وبالتالي فإن ناتج الطباعة سيكون :

3 6 2 7

والآن يجب أن نلاحظ أن المؤثر ++ هو مؤثر أحادى

unary operator

أي أنه يؤثر على قيمة واحدة (حيث يدخل على قيمة واحدة فيضيف إليها 1) . وكذلك فإن المؤثر -- هو الآخر مؤثر أحادى (حيث يدخل على قيمة واحدة فيطرح منها 1). ولكن بالإمكان وضع هاذين المؤثرين على يمين المتغير أو على شماله على النحو :

count++ أو ++count

يوصف ++ عندما يكون على يمين المتغير بأنه مؤثر بَعدى postfix ، وعندما يكون على الشمال بأنه مؤثر قبلي prefix .

ما الفرق بين الحالتين ؟ الفرق هو في أسبقية العمليات، حيث المؤثر البعدي يحدث تأثيره بعد الانتهاء من تنفيذ الجملة الوارد بها ، أما المؤثر القبلي فيحدث تأثيره قبل العمليات الأخرى في نفس الجملة.

فمثلاً الجمل :

```
k = 1  ;
printf ( " %d " , k++);
```

تطبع الناتج 1 لأن الطباعة تتم قبل إضافة 1 إلى k .

أما الجمل :

```
k = 1   ;
printf ( " %d " ,++k)  ;
```

فتطبع 2 حيث يضاف 1 إلى k قبل الطباعة.

لاحظ أن الجملة

```
k++   ;
```

تكافئ الجملة

++k ;

حيث لا يوجد بها إلا أمر إضافة 1 إلى k .

الآن وقد درسنا استخدام مؤثرات التغيير ، دعنا نعيد كتابة البرنامج بالشكل (5.4.4) لحساب المتسلسلة :

$$1+1/2! + 1/3! + \dots + 1/n!$$

حيث يمكننا الآن كتابة ++k أو k++ بدلا من (k = k+ 1) ، و أيضاً

sum + = t ;

بدلا من

sum = sum + t ;

وكذلك يمكننا أن نكتب

t /= k ;

بدلاً من

t = t / k

هذه الطرق المختصرة المستخدمة في البرنامج بالشكل (5.6.2) تعطـــى طابعاً مميزاً لبرامج لغة سى .

```
main()
{  int k,n ;
   float t, sum;
   printf("\n Please enter n-->");
   scanf("%d",&n);
   for( k=1 , t=1 , sum=0;  k<= n;   k++)
   {
     t /= k;
     sum += t;
   }
   printf("\n sum =%f",sum);
}
```

الشكل (5.6.2) برنامج حساب المتسلسلة باستخدام مؤثرات التغيير

5.7 الحلقات المتداخلة Nested loops

من الممكن أن تحتوى حلقة for على حلقة for أخرى على النحو التالي :

```
for  ( i = 1 ; i < = n ;  i++ )
     {……
          for ( k = 1 ; k < = m , k++)
            {  ....
           }
        }
```

وهوما يعرف بالحلقات المتداخلة (أي وجود حلقة داخل حلقة).

مثال : اكتب برنامجا لقراءة درجة الامتحان النصفي والامتحان النهائي لعدد N من الطلبة، وإيجاد مجموع درجات كل طالب والمتوسط العام، علماً بأن درجة الامتحان النصفي من 40، والامتحان النهائي من 60 ، ويجب أن يقوم البرنامج بالتنبيه إلى الخطأ إذا تجاوزت درجة طالب الحد الأقصى في أحد الامتحانين.

سوف نستخدم في هذا البرنامج المتغيرات التالية:

- درجة الامتحان النصفي midterm
- درجة الامتحان النهائي final
- المجموع total

للتحقق من إدخال درجة الامتحان النصفي نستخدم حلقة لانهائية لا خروج منها إلا بإدخال درجة أقل من أو تساوى 40 . كما نحتاج إلى حلقة مماثلة لإدخال درجة صحيحة للامتحان النهائي أقل من أو تساوى 60.

وهذا المثال يبين الطريقة المستخدمة غالبا في التطبيقات العملية للتحقق من سلامة البيانات، إذ لا يقبل البرنامج البيانات إلا إذا كانت في حدود المدى المحدد لها. ويكون ذلك باستخدام حلقة لا نهائية عند قراءة البيانات بحيث – كما ذكرنا– لا يتم الخروج من الحلقة إلا إذا تم إدخال البيانات بشكل مقبول. وقد يعمل المبرمج عدادا يحسب عدد المحاولات لإدخال البيانات بشكل سليم، وإذا تجاوز المستخدم العدد المحدد للعداد يتم الخروج من الحلقة وإيقاف تنفيذ البرنامج .

```
main()
{  float midterm, final, total;
   int k,n;
   printf("\n Enter number of students-->");
   scanf("%d",&n);
   for(k=1; k<=n ; k++)
   {       for(;;)
           { printf("\nPlease enter midterm-->");
             scanf("%f",&midterm);
             if(midterm<= 40) break;
             else    printf("\n data entry error");
           }
           for(;;)
           { printf("\nPlease enter final-->");
             scanf("%f",&final);
             if( final <= 60 ) break;
             else    printf("\n data entry error");
           }
           total = midterm + final;
           printf ("\n
            total grade of student number %d is %5.2f",
                                    k,total);
   }
}
```

الشكل (5.7.1) الحلقات المتداخلة

5.8 التكرار باستخدام goto

أخيرا لابد من الإشارة إلى جملة goto التي لا تكاد تخلو منها أي لغة برمجة، وإن كانت غير مرغوبة من قبل المنادين بالبرمجة الهيكلية structured programming نظرا لسوء استخدامها بكثرة خاصة للمبرمجين المبتدئين ، حيث يجدون فيها وسيلة سهلة في كتابة البرامج التي تحتوي على حلقات، ولكن كثيرا ما يقعون في مشكلة غموض البرنامج وصعوبة تتبعه. لذلك لا ننصح باستعمالها إلا للضرورة.

قبل أن ندرس الصورة العامة لاستخدام goto يجب أن نشير إلى الجملة في لغة سي يمكن أن تحتوي على عنوان label يميزها عن غيرها من الجمل في الدالة الواحدة.

فمثلا الجملة:

```
sum = sum + cost;
```

يمكن أن يكتب معها عنوان على النحو:

```
LB1 : sum + cost ;
```

حيث LB1 هو العنوان الذي انتقيناه لهذه الجملة. لاحظ ضرورة وضع الشارحة : بين الجملة وعنوانها.

الآن يمكننا أن نحول تنفيذ البرنامج إلى هذه الجملة بواسطة goto على النحو التالي:

goto LB1

أي أن الصورة العامة لجملة goto هي :

goto LABEL;

حيث LABEL يرمز لعنوان الجملة المطلوب التحول إليها.

مثال (5.8.1)

أعد كتابة البرنامج (5.2.3) لحساب مجموع الأعداد من 1 إلى 100 وذلك باستخدام goto بدلا من while.

```
main()
{    int sum=0 , k=1;
  LB1  :  sum += k;
        k++;
        if(k<=100)
                goto LB1;
        else
                printf("\n total=%d");
}
```

الشكل (5.8.1) برنامج يستخدم goto

5.9 تمارين

في المسائل التالية ، اكتب البرنامج بثلاثة طرق مستخدما

ا . دورة طالما while
ب . دورة do - while
ج . دورة for

اعتبر أن n هو عدد صحيح يتم إدخاله .

1 . برنامج لطباعة الأعداد الفردية من 1 إلى n.
2 . برنامج لإيجاد مجموع الأعداد الزوجية من 10 إلى n.
3 . برنامج لإيجاد حاصل ضرب الأعداد من 5 إلى n ، أي

$$5 * 6 * 7 * \; .... \; * (n-1) * n$$

4 . برنامج لإيجاد المجموع

$$1 + 1/2 + 1/4 + 1/8 + \; .... \; + 1/2^n$$

5 . برنامج لإيجاد أعلى درجة وأقل درجة من بين درجات n من الطلبة، والفرق بينهما.

6 . برنامج لحساب مجموع ما تصرفه أسرة في أسبوع، علماً بأن المدخلات هي ما تصرفه الأسرة في اليوم، وهي :
المواد الغذائية food والملابس cloths والمواصلات car وغير ذلك other ، أي أن المصروف اليومي daily هو :

daily = food + cloths + car + other

7 . برنامج لحساب المتسلسلة

$$1/2! + 1/4! + 1/6! + \ldots + 1/n!$$

8 . برنامج لحساب المتسلسلة

$$1 - 1/2 + 1/3 - 1/4 + 1/5 + \ldots\ 1/n$$

9 . استخدم طريقة الحلقة اللانهائية للتحقق من إدخال قيمة x بالشكل الصحيح وهو

$$5 \leq x \leq 30$$

10 . اكتب برنامجا لطباعة المقابل بالنظام الثماني والسادس عشري للأعداد من 1 إلى 100.

11 . لديك جدول من n صف و m عمود حيث تحتوى كل خانة من خانات الجدول على عدد صحيح. اكتب برنامجا لحساب وطباعة مجموع كل صف.

12 . أعد البرنامج بتمرين (11) لحساب وطباعة أكبر قيمة في كل صف .

13 . اكتب برنامجا لطباعة جدول الضرب للأعداد من 0 إلى 9 .

14 . ما هي القيم التي يطبعها البرنامج التالي ؟

```
main ( )
{
     int  x = 1 , y = 2 , z = 3  ;
      x  + = y+ +;
     z   * =  5  ;
     printf ( " \ n   %d  %d  " , x , y , z++) ;
}
```

6

الباب السادس

المصفوفات و النضائد

Arrays & Strings

6.1 مقدمة

المصفوفة هي مجموعة من العناصر المتجانسة تحت اسم واحد، حيث يمكن تمييز كل عنصر بترتيبه (أي دليله) في المصفوفة . فمثلاً مجموعة الأعداد :

$a_0 = 5.1$
$a_1 = 6.3$
$a_2 = 7.4$
....
$a_9 = 8.2$

تكوّن مصفوفة من 10 عناصر من النوع الكسري float .

افترض أننا نريد قراءة هذه العناصر وإيجاد متوسطها ثم إيجاد الفرق بين كل عنصر والمتوسط.

ستوضح لنا هذه المسألة أهمية المصفوفات . إذا اعتبرنا الطريقة العادية، حيث نرمز للمدخلات بمتغير واحد وليكن x ، فإن حساب المتوسط يتم بسهولة على النحو التالي :

```
 for( k = 1 , sum = 0  ; k < =10 )
{  scanf ( "%f "  , &x )  ;
   sum = sum +  x  ;
}
average  =  sum / 10  ;
```

ولكن الآن لحساب الفرق بين كل عنصر والمتوسط ، نجد أن علينا قراءة البيانات مرة أخرى، حيث لا يوجد لدينا في ذاكرة الحاسوب إلا آخر قيمة تمت قراءتها للمتغير x .

لذلك فإن الفكرة الأساسية وراء مفهوم المصفوفة array هي تخزين كل عناصر المصفوفة في الذاكرة الرئيسية والرجوع إليها وقت الحاجة .

هذا يتطلب طبعاً التمييز بين عنصر وآخر، نظراً لأن جميع العناصر تندرج تحت اسم واحد هو اسم المصفوفة ، فإن كل عنصر يتميز برقم صحيح يرمز

لترتيبه في المصفوفة، ويسمى هذا الرقم بالدليل index. فمثلاً العنصر v [i] هو العنصر رقم i في المصفوفة v .

6.2 الحجز في الذاكرة

إذا أردنا استخدام مصفوفة في البرنامج ، فلابد في البداية من حجز عدد من المواقع في الذاكرة يكفى لجميع عناصر هذه المصفوفة، ومع الحجز يجب الإعلان عن نوع هذه العناصر حتى تخصص لها المواقع المناسبة لحجمها .
فإذا كانت المصفوفة cost ذات 7 عناصر من النوع الكسري ، فإن الإعلان عن ذلك يتم على النحو التالي:

```
float cost [7] ;
```

لاحظ أن ذلك يعنى أن عناصر المصفوفة هي :

```
cost [0]
cost [1]
cost [2]
cost [3]
cost [4]
cost [5]
cost [6]
```

أي أن آخر عنصر يكون دليله أقل بواحد من الرقم المعلن عنه في الحجز ، والسبب هو البداية من الصفر ، أي أن العنصر الأول دليله = 0 ، والعنصر الثاني دليله = 1 ، .... الخ .

مثال : اكتب برنامجا لحساب متوسط دخل فرد خلال أسبوع (أي سبعة أيام) من دخله اليومي، ثم طباعة الفرق بين دخل كل يوم والمتوسط .

سبق وأن تطرقنا إلى هذا المثال في المقدمة للمصفوفات ، ورأينا أن عدم استخدام المصفوفات يؤدى إلى ضرورة قراءة البيانات مرتين .
طبعاً يمكننا في هذا المثال تخزين دخل الفرد في كل يوم على النحو :

income0 , income1 , income2 , ..., income 6

أي بدون استخدام المصفوفات، ولكن ماذا لو كان المطلوب المتوسط الشهري أو حتى السنوي ؟ بكل تأكيد ستكون قائمة الأسماء طويلة جدا.
لذلك نستخدم الإعلان

```
float income [7] ;
```

لنعلن عن مصفوفة ذات 7 عناصر كسرية اسمها income .
بعد قراءة هذه العناصر وحساب المتوسط نستطيع الرجوع إليها لحساب الفرق diff بين كل عنصر والمتوسط كما هو مبين بالبرنامج (6.2.1) .

```
main()
{       float income[7], sum , average;
        int i;
        for(i=0; i<=6 ; i++)
        {
          printf("\n Enter income for day[%d]"
                          ,i+1 );
          scanf("%f",&income[i]);
          sum += income[i];
        }
        average=sum/7;
        printf("\n avrage=%6.3f",average);
        printf("\n day        income         diff");
        for(i=0; i<=6 ; i++)
          printf("\n %d      %6.2f  %6.2f",
        i+1,income[i],income[i]-average);
}
```

الشكل (6.2.1) برنامج حساب المتوسط والفرق بين كل عنصر والمتوسط

6.3 ترتيب المصفوفة Array Sorting

من أهم تطبيقات المصفوفة أنها تمكننا من إعادة ترتيب عناصرها إمـــا ترتيبـــاً تنازلياً أو ترتيباً تصاعدياً .

هناك طبعاً عدة طرق لترتيب المصفوفات. مثلا لو أعطيت الأعداد التالية :

x [0] = 10
x [1] = 42
x [2] = 33
x [3] = 15
x [4] = 26

كيف ترتبها تنازلياً (أي ابتداء من أكبر قيمة إلى أصغر قيمة) ؟ هناك طبعا عدة طرق لاجراء هذا الترتيب. إحدى هذه الطرق مثلاً الخوارزمية التالية :

- من i=0 إلى i = n-2 (حيث n-1 هو دليل آخرعنصر في المصفوفة)

  إذا كانت x [i] < x [i+1] فاستبدل قيمتيهما .

- هل أصبحت المصفوفة مرتبة ؟

- إذا كانت الإجابة نعم توقف وإلا فارجع إلى الخطوة ❶ .

والبرنامج التالي يترجم هذه الخوارزمية إلى لغة سي:

```
/*----------------PROGRAM EX631.C -----------*/
#define N 5
main()
{       int x[N], i ,k, temp , sorted;
```

```
	for(i=0; i<N ; i++)
	{
		printf("\nEnter element[%d] ",i);
		scanf("%d",&x[i]);
	}
	for(k=0;;k++)
	{	sorted=1;
		for(i=0; i<N-1 ; i++)
		{
			if(x[i+1] > x[i] )
			{
				sorted = 0;
				temp = x[i];
				x[i] = x[i+1];
				x[i+1] = temp;
			}
		}
		if(sorted) break;
	}
	printf("\n Sorted array after %d iter", k);
	for(i=0; i<N ; i++)
		printf("\n %d",x[i]);
}
```

الشكل (6.3.1) ترتيب البيانات

من الواضح في هذا البرنامج أن هناك حلقتين . الحلقة الخارجية تستمر بـــدون تحديد حتى يتم ترتيب المصفوفة ، والحلقة الداخلية تتم بعد n دورة.
من الملاحظ أيضا في البرنامج بالشكل (6.3.1) استخدام المتغير sorted الذي يبدأ بالقيمة 1 ثم يتحول إلى 0 إذا ما وجدنا عنصرين غير مرتبين ، وفى نهاية الدورة الداخلية نختبر قيمة هذا المتغير ، فإذا كانت 1 فذلك يعنـــى أن عمليـــة الترتيب قد تمت كما يجب . ونخرج من الدورة الخارجية اللانهائية بالجملة :

```
if ( sorted ) break ;
```

وهى تكافئ الجملة :

```
if ( sorted = = 1) break ;
```

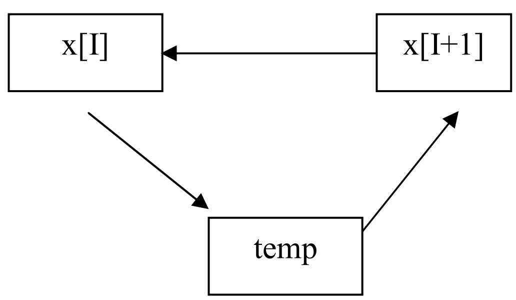
ولكن أكثر اختصاراً .

وثمة ملاحظة أخرى جانبية عن هذا البرنامج وهي أنه يطبع عـــدد الـــدورات iterations التي أداها للوصول إلى حل (أي إلى المصفوفة المرتبة تنازليـــا) . طبعاً هذا العدد يختلف حسب المصفوفة المدخلة ، ولكنه في أســـوأ الأحـــوال لا يزيد عن N (عدد العناصر) .
كان بالإمكان أن نكتب الدورة الخارجية على النحو :

```
for( k=0 ; k<N ; k++) { ……}
```

والاستغناء عن اختبار المتغير sorted ، والحلقة اللانهائية، ولكن قد لا نحتاج أحياناً إلى N من الدورات بل أقل من ذلك ، وبالذات إذا كانت المصفوفة المدخلة تكاد تكون مرتبة تنازليا ، لذلك استخدمنا المتغير sorted للخروج من الدورة الخارجية بمجرد الوصول إلى المصفوفة المرتبة.

لاحظ أيضاً أن البرنامج يقوم بعملية استبدال x [i] مع x [i+1] بالطريقة المعتادة في البرمجة ، وهى تتم عن طريق وسيط مؤقت temp نضع فيه x [i] قبل وضع x[i+1] في x [i] على الشكل التالي :



الشكل (6.3.1) استبدال x [i] مع x [i+1]

6.4 النضائد Strings

النضيد هو عبارة عن مصفوفة من الرموز . وهذه الرموز تشمل الحروف الهجائية ، والأرقام والإشارات ، المبينة بجدول آسكى ascii (أنظر الملحق) .
في لغة سى ، يوضع النضيد بين علامات التنصيص المزدوجة مثل :

" Kamal Ahmed "
" 32 Cairo Street "

لاحظ أن هذه الرموز (سواء أكانت حروف أم أرقاماً) تخزن في ذاكرة الحاسوب كأرقام ثنائية وذلك حسب الشفرة المستخدمة، لذلك يمكن أن ننظر إلى النضيد على أنه مصفوفة من الأعداد الصحيحة. وهذه الأعداد لا يتجاوز كل منها العدد الذي يحمله الحيز المخصص لحرف واحد في الحاسوب ، وهو عادة بايت واحدة أي 8 بت. عند تخصيص حيز في الذاكرة للنضيد يجب الانتباه إلى إضافة موقع واحد لرمز نهاية النضيد (ويرمز له بالرمز '\0') ، فإذا كان النضيد المطلوب هو :

"UNIVERSITY"

فيجب الإعلان عن 11 رمز لهذا النضيد (10 أحرف بالإضافة إلى رمز الانتهاء) ، ويكون الإعلان على النحو التالي :

```
char w [11] ;
```

حيث يرمز المتغير w لمصفوفة عناصرها من النوع الرمزي char ذات 11 عنصر منها 10 رموز عادية ورمز الانتهاء '\0'

مثال : اكتب برنامجا يقوم بقراءة اسم المستخدم ، ولقبه (بحيث لا يزيد الاسم عن 11 حرفا ولا يزيد اللقب عن 14 حرفا) ويطبع الآتي:

Your name is ....

حيث يطبع الاسم واللقب في هذا السطر مع وضع فراغ واحد بينهما.

لدينا هنا مصفوفتان الأولى للاسم ولتكن name، والأخرى للقب ولتكن family . الأولى تحتاج إلى 12 رمزاً (11 حرفاً مع رمز الانتهاء) والثانية تحتاج إلى 15 رمزاً (14 حرفاً مع رمز الانتهاء) .

```
main()
{
        char name[12];
        float family[15];
        printf("\n What is your first name? ");
        scanf("%s",name);
        printf("\n What is your family name?" );
        scanf("%s",family);
        printf("\n Your name is %s %s ", name, family);
}
```

الشكل (6.4.1) قراءة نضيدين وطباعتهما في سطر واحد

ملاحظات

❶ لاحظ الآن أن دالة القراءة scanf تستخدم الرمز s% لقراءة النضيد (حيث s ترمز لكلمة string نضيد) .

❷ لاحظ أيضاً عند قراءة نضيد ما إمكانية الاستغناء عن الرمز & الذي يوضع عادة قبل المتغير المطلوب قراءته في دالة scanf ، وسيتضح السبب فيما بعد .

❸ توجد في لغة سى دالة أخرى غير دالة scanf لقراءة النضيد وهى gets (اختصار get string) ، أي كان بالإمكان كتابة الجملة :

```
gets ( name ) ;
```

بدلًا من الجملة:

```
scanf ( " %s " , name ) ;
```

كما في المثال التالي:

<u>مثال</u> : اكتب برنامجا لقراءة نضيدين string2 و string1 لا يزيد طول كل منهما عن 20 رمزاً، وتكوين نضيد ثالث string3 يتكون من دمج النضيدين بحيث يأتي string2 في نهاية string1 .

يبين الشكل (6.4.2) البرنامج المطلوب وفيه نلاحظ ما يلي:

1 – استخدم الدالة gets لقراءة نضيد

2– استخدام الاختبار

string [i] = = ' \0 '

لاختبار نهاية النضيد string

3- بعد وضع النضيدين في النضيد string3 يجب إنهاء النضيد الجديد بوضع '\0' في نهايته ، بواسطة الجملة :

string3 [k] = ' \0 ' ;

4- إمكانية طباعة نضيد بواسطة الدالة puts مثل :

puts (string3) ;

5- عملية ربط نضيدين في نضيد واحد من المسائل المعروفة، ويطلق عليها عادة مصطلح concatenation، ويوجد لها في لغة سى دالة جاهزة اسمها strcat .

لذلك كان من الممكن اختصار البرنامج (6.4.2) باستخدام هذه الدالة على النحو التالي:

strcat (string1 , string2) ;

وهى جملة تقوم بإضافة string2 إلى string1 .

```
main()
```

```
{
        char string1[20] , string2[20] , string3[40] ;
        int i , k,j ;
        printf("\n enter string1-->");
        gets( string1 );
        printf("\n enter string2-->");
        gets( string2 );
        for(i=0; ; i++)
        {       if( string1[i] == '\0' ) break;
                string3[i]=string1[i];
        }
        for(k=i, j=0 ;   ; k++,j++)
        {       if( string2[j] == '\0' ) break;
                string3[k] = string2[j];
        }
        string3[k] = '\0' ;
        puts ( string3 );
}
```

الشكل (6.4.2) برنامج دمج نضيدين

وإذا أردنا نسخ string1 في string3 نستخدم الدالة strcpy (الكلمة مصدرها string copy) على النحو التالي :

```
strcpy ( string 3 , string 1)   ;
```

مثال : اكتب برنامجا لقراءة عدد صحيح يتكون من 5 خانات وطباعته مع التأكد من عدم وجود خطأ في الإدخال . أي أن البرنامج لا يقبل العدد المدخل حتى يتأكد من عدم وجود رمز غير الأرقام من 0 إلى 9 .

سوف نتبع في هذا البرنامج الخوارزمية التالية :

1. اقرأ الرقم الأول كرمز (دون إظهاره على الشاشة).
2. هل تم إدخال الرقم بصورة صحيحة ؟ أي هل يقع الرمز في الفئة ؟
   { 0 , 1 , 2 , ... , 9 }
3. إذا كانت الإجابة (نعم) اكتب الرقم المدخل على الشاشة ، وانتقل إلى الرقم الذي يليه (إلى غاية 5 أرقام) .
   وإلا ستستمر الحلقة لانهائية مع إصدار تنبيه صوتي بذلك.

هذه الخوارزمية نترجمها إلى برنامج بلغة سى في الشكل (6.4.3)، حيث نلاحظ ما يلي:

- الدالة getch (لاختصار get character) تقوم باستقبال رمز واحد من لوحة المفاتيح دون إظهاره على الشاشة.

- وبمناسبة ذكر هذه الدالة ، نلاحظ أن هناك دالة أخـــرى تـــؤدى نفـــس الغرض ولكن مع إظهار الرمز المدخل على الشاشة وهذه الدالة هـــي getche حيث تزيد بحرف e على الدالة getch.
- الدالة putchar (اختصار put character) تقوم بإظهار رمز واحـــد على الشاشة .

```
#include <stdio.h>
main()
{
        char x[5];
        int i;
        printf("\n Please enter a number-->");
        for(i=0;i<=4 ;i++)
                for(;;)
                {       x[i]=getch();
                        if( x[i]>= '0' && x[i]<= '9')
                        {   putchar(x[i]);
                                break;
                        }
                        else
                        printf("\a");
                }
        x[5]='\0';
        printf("\n The number entered is %s ", x);
        getch();
}
```

الشكل (6.4.3) التحقق من إدخال عدد صحيح

- بعد أن عينّا الخانات الخمس في العدد المطلوب وضعنا علامة نهاية النضيد في الخانة السادسة ، أي x [5] ، على النحو :

```
x [5] = '\0' ;
```

وذلك لغرض اكتمال النضيد x وطباعته باستخدام الدالة printf . كان من الممكن أيضاً طباعته بواسطة الدالة puts .

- نلاحظ أن الدالة getch التي وضعت في آخر البرنامج تهدف إلى الاحتفاظ بشاشة الإخراج حتى يتم إدخال أي رمز في لوحة المفاتيح keyboard .
- أخيراً نلاحظ ضرورة وضع التوجيه

```
# include < stdio.h >
```

قبل الدالة main ( ) نظراً لاستخدام الدالتين :

getch ( ) و putchar ( )

الجملة ; (" a\ ") printf تقوم بإصدار صوت الجرس bell تنبيها للخطأ.

6.5 المصفوفات ذات البعدين 2- dimensional arrays

تــتكون المصفوفة ذات البعدين من عدد من الصفوف وعدد من الأعمدة. فمثلا البيانات التالية والتي تخص عدد الطلبة في 4 فصول دراسية و 3 أقسام بمعهــد عالٍ ، يمكن تمثيلها بمصفوفة ذات بعدين :

الفصل الدراسي	1	2	3	4
طلبة الحاسوب	78	56	42	35
طلبة الرياضة	58	47	61	38
طلبة الاحصاء	45	36	25	19

الشكل (6.5.1) أعداد الطلبة حسب الفصل الدراسي والتخصص

إذا رمزنا لهذه المصفوفة بالاسم students ، يمكننا الإعلان عنها في لغة سى على الصورة :

```
int  students [3][4]  ;
```

حيث عدد الأعمدة =4 و عدد الصفوف=3 .

ومعنى ذلك أن students [0][0] تعنى عدد طلبة الحاسوب (computer) في الفصل الأول ، أما students[1][1] فتعنى عدد طلبة الرياضة (Math) فــي الفصل الثاني ، وهكذا .....

مثال : اكتب برنامجا لحساب وطباعة :

أ. مجموع عدد الطلبة في كل قسم.
ب. مجموع عدد الطلبة في كل فصل .

علماً بأن المعطيات كما هو مبين بالشكل (6.5.1) .

البرنامج المطلوب مبين بالشكل (6.5.2) ، دعنا نلقى نظرة فاحصة على هـذا البرنامج، لتستوقفنا النقاط التالية :

```
#define N 3
#define M 4
main()
{
        int  students[3][4]
                ={78,58,45,56,47,36,42,61,25,35,38,19}
                , dept[3], sem[4],    i, j;
        char format[10]=" %d\t";
        for ( i=0 ; i<N ; i++)
        {       dept[i]=0;
                for(j=0; j<M ; j++)
                        dept[i] += students[i][j];
        }
```

```
        for(j=0; j < M ; j++)
        {       sem[j]=0;
                for(i=0 ; i<N ; i++)         sem[j] += students[i][j];
        }
        for(i=0 ; i<N; i++)
        {       printf("\n") ;
                for(j=0 ; j<M; j++)
                        printf(format, students[i][j])  ;
                printf(" %d",dept[i]);
        }
        printf("\n");
        for(j=0; j<M ; j++)         printf(format,sem[j]);
}
```

الشكل(6.5.2) برنامج معالجة مصفوفة ذات بعدين

① نلاحظ أنه لا توجد دالة الإدخال scanf في البرنامج ، ولكن تم وضع الجدول المطلوب داخل البرنامج عند الإعلان عن المصفوفة students على النحو التالي:

```
int studens [3][4] = { 78 , 58 , 45 , 56 , 47 , 36 ,
                       42 , 61 , 25 , 35 , 38 , 19 }
```

حيث وضعنا الجدول على شكل قائمة أفقية ابتداء من العمود الأول في الجدول ، يليه العمود الثاني ، .... وهكذا .

تسمى هذه الطريقة بتخصيص القيم الابتدائية للمصفوفة array initialization .

ويمكن استخدامها في المصفوفة الأحادية وثنائية البعد على حد سواء . على سبيل المثال الجملة :

```
char  format [10]  = " %d \ t "   ;
```

تقوم بالإعلان عن أن المتغير format هو من نوع النضيد ، وتخصص لـــه القيمة الابتدائية " %d \ t " .

② الوصف " %d \ t " يعنى طباعة عدد صحيح ثم ترك بعض الفراغـــات نظراً لوجود الرمز t \ الـــذي يستخدم في طباعة الجداول tab . أي أن تأثير t \ هو الانتقال إلى حقل جديد في نفس السطر.

وبالمناسبة فإن الرموز التالية تستخدم عادة في عملية الوصف :

الرمز	المعنى
\a	الجرس bell
\b	الرجوع إلى الـــوراء خانـــة واحـــدة back space
\n	الانتقال إلى سطر جديد
\t	الانتقال إلى حقل جديد في نفس السطر

6.6 ترتيب الأسماء

نظراً لما لعملية ترتيب قوائم الأسماء ترتيبا أبجديا من أهمية بالغة في التطبيقات الإدارية، نخصص لها هنا موضوعا كاملا.
نحتاج طبعا عند ترتيب الأسماء إلى مقارنتها أبجديا. فإذا قارنا مثلا بين
الاسم " Ahmed" والاسم " Ali "
نجد أن

"Ali " > "Ahmed "

رغم أنهما يتساويان في الحرف الأول ، إلا أن الحرف الثاني من "Ali" يأتي في الترتيب بعد الحرف الثاني من " Ahmed" .
تفيدنا في هذه المقارنة الدالة strcmp التي تقوم بمقارنة نضيدين ، ويأتي هذا الاسم من : string comparison ، وهى تعمل على النحو التالي:

$$\text{strcmp}(\text{s1}, \text{s2}) = \begin{cases} negative & if \quad s1 < s2 \\ 0 & if \quad s1 = s2 \\ positive & if \quad s1 > s2 \end{cases}$$

وبما أننا نريد أن يكون الترتيب تصاعديا (أي من الأسماء التي تبدأ بحرف A إلى الأسماء التي تبدأ بحرف Z) سيكون الاختبار على النحو التالي:

```
if  ( strcmp (name[i+1] , name[i] )  <  0  )
        {  sorted = 0   ;
          strcpy  ( temp , name [i] )  ;
          strcpy (name [i]  ,  name [i+1] )  ;
          strcpy ( name [i+1] , temp )  ;
        }
```

نلاحظ هنا استغلال الدالة strcpy في نسخ نضيد إلى آخر، حيث لا يجوز في لغة سى أن نكتب مثلاً :

```
temp =  name [i]  ;
```

لأن temp و name [i] من نوع النضيد string .

اتبعنا في هذا البرنامج نفس الخوارزمية التي اتبعناها في البرنامج بالشكل (6.3.1) مع ملاحظة الآتي :

❶ استخدام المصفوفة ثنائية البعد name[N][L]، حيث N عدد الأسماء بالقائمة و L طول كل اسم بالحرف .

❷ استخدام الدالة strcpy لنسخ نضيد إلى آخر .

❸ استخدام الدالة strcmp لمقارنة نضيدين .

```
#define  N 5
#define  L 12
main()
{       char name[N][L], temp[L] ;
        int i ,j, k  , sorted;
        for(i=0; i<N ; i++)
        {
                printf("\nenter name[%d] ",i);
                scanf("%s",&name[i]);
        }
        for(k=0; k<N; k++)
        {      sorted=1;
                for(i=0; i<N-1 ; i++)
                {
                 if(strcmp(name[i+1] , name[i])<0 )
                        {       sorted = 0;
                                strcpy(temp, name[i] );
                                strcpy(name[i] , name[i+1]);
                                strcpy(name[i+1] , temp);
                        }
                }
                if(sorted) break;
        }
        printf("\n Here is the sorted array after %d iterations",k);
        for(i=0; i<N ; i++)
                printf("\n %s",name[i]);
}
```

الشكل (6.6.1) برنامج ترتيب الأسماء

مثال : إضافة اسم إلى قائمة مرتبة

نفرض أن لدينا المصفوفة old_array مرتبة ترتيباً أبجديا تصاعديا، و أن المطلوب هو كتابة برنامج لإضافة اسم ووضعه في المكان المناسب من القائمة الجديدة new_array .

تسمى هذه العملية بالإدراج inserting ، وهى تتطلب (بعد تحديد المكان المناسب للاسم الجديد) إزاحة كل الأسماء التي تليه في القائمة. فمثلا إذا كانت القئمة القديمة مرتبة على النحو التالي:

i	Old Array
0	Anas
1	Huda
2	Lubna
3	Omar
4	Suad

.وأضفنا إليها الاسم Shada فإن القائمة الجديدة يجب أن تكون على النحو التالي:

i	New Array
0	Anas

1 Huda
2 Lubna
3 Omar
4 Shada
5 Suad

و لتحديد الموقع (وليكن m) للاسم المضاف new_name ، نلاحظ أن :

old_array [m] < new_name < old_array [m+1]

وهذا هو الشرط الذي نستخدمه في البرنامج لتحديد قيمة m .

والآن يـــتضح أن المصفوفة الجديدة new_array يمكن تكوينها كما يلى :

```
if i < m    new_array[i] = old_array[i];
if i == m   new_array[i] = new_name;
if i > m    new_array[i] = new_array[i-1];
```

حيث N .... , 3 , 2 , 1 , 0 =i

والشكل (6.6.3) يبين البرنامج المطلوب ، مع ملاحظة الآتي :

المصفوفة الجديدة new_array تزيد عن المصفوفة القديمة old_array مـــن حيث الحجم بموقع واحد ، وبالتالي فإن الإعلان عن المصفوفتين يتم على النحو التالي:

```
char old_array [N][L] , new_array [N+1][L] , new_name[L] ;
```

لتحديد موقع new_name نجرى الاختبار:

old_array[i] < new_name < old_array[i+1]

```
#define  L 12
#define  N 5
main()
{       char old_array[N][L], new_array[N+1][L] ,
        new_name[L];
        int i ,m  ;
        for(i=0; i<N ; i++)
        {
                printf("\nenter name[%d] ",i);
                scanf("%s",&old_array[i]);
        }
        printf("\nPlease enter new name → ") ;
        scanf("%s",new_name);
        for(i=0;i<N; i++)
                if( strcmp( new_name , old_array[i]) > 0 &&
                   strcmp( new_name , old_array[i+1]) < 0 )
                {  m=i+1 ;  break ;  }
        for(i=0; i<=N ; i++)
    { if(i<m)
           strcpy(new_array[i],old_array[i]);
          if(i==m)
```

```
          strcpy(new_array[i], new_name);
        if(i>m)
       strcpy(new_array[i], old_array[i-1]);
       }
      printf("\n Here is the new array ");
      for(i=0; i<=N ; i++) printf("\n %s",new_array[i]);
}
```

الشكل(6.6.3) إضافة اسم إلى قائمة مرتبة.

ويكتب في لغة سى على النحو التالي :

```
strcmp ( new_name  , old_array [i] )  >  0
&&  strcmp (new_name  ,  old_array [i+1] )  <  0
```

نلاحظ أن إضافة اسم واحد للمصفوفة أدت إلى تكوين مصفوفة أخرى جديـــدة ، وهذا يعتبر إسرافاً في استغلال الذاكرة . لذلك نستخدم ما يعرف عادة بـــالقوائم المرتبطة linked lists التي تعتمد على مفهوم المؤشرات pointers وهو ما سندرسه فيما بعد .

كما نلاحظ أن المصفوفة old_array والمصفوفة new_array ذواتا بعد ثابت ، يتم تحديده في جملة الإعلان عن النوع والبعد ، في بداية البرنامج . ولكن من الناحية العملية من الأفضل أن تكون هناك إمكانية تغيير حجم المصفوفة (أي بُعد المصفوفة) حسب مقتضيات الأمر. فقد نحجز في الذاكرة مواقـــع أكثـــر ممـــا

نحتاج، مما يسبب ضياع مواقع التخزين ، أو قد نحجز أقل مما نحتاج مما قــد يسبب فشل البرنامج . هذه المشكلة أيضاً يتم حلها بواسطة المؤشــرات ، التـــي نتناولها في باب خاص.

6.7 تمارين

1. اكتب برنامجا لحساب مجموع مربعات الأخطاء

$$s = \sum (x_I - xbar)^2$$

حيث $\sum$ تعني الجمع من I=0 إلى I=n-1. أما xbar فهي ترمز لمتوسط قيم x_I.

افترض أي قيمة صحيحة للمتغير n.

لماذا نحتاج إلى استخدام المصفوفة في هذا البرنامج ؟

2 . المعادلة

$$y_{i+1} = (1+r)\, y_i$$

تصف الرصيد في المصرف بعد مرور i من السنوات، حيث r يمثـــل النـــسبة المئوية للربح. اكتب برنامجا لحساب :

أ. الرصيد بعد 10 سنوات.
ب. مقدار الربح بعد 10 سنوات.
ج. عدد السنوات اللازمة حتى يفوق الربح الرصيد الابتدائي.

اعتبر أن الرصيد الابتدائي هو 1000 وأن r = 0.05 .

3. اكتب برنامجا لقراءة مصفوفة من 10 عناصر عددية صحيحة، و اختبار ما إذا كانت المصفوفة مرتبة تصاعدياً .

4. اكتب برنامجا يقوم بقراءة عدد كسري ويتحقق من عدم وجود خطـــأ فـــي الإدخال وذلك بعدم قبول أي رمز غير الأرقام من 0 إلى 9 والنقطة العشرية.

5 . اكتب برنامجا يقوم بقراءة نضيد مكتوب بالحروف الصغيرة

a , b , c , ... z

وتحويلها إلى حروف كبيرة

A , B , C , ... Z

لاحظ أن الفرق بين الحرف الصغير ومقابله من الأحرف الكبيرة هو مقدار ثابت (في جدول أسكى) وهو 32. فمثلاً رقم c في جدول أسكى هو 99، بينما رقم C هو 67.

6. لديك مجموعة من الطلبة ودرجاتهم في مقرر البرمجة. والمطلوب ترتيب أسمائهم حسب درجاتهم ، أي أول اسم في القائمة هو صاحب أعلى درجة ، وهكذا ..

7 . إذا كانت البيانات x_i مرتبة تنازلياً أو تصاعديا ، وكان عددها فرديا ، فإن القيمة الوسطى تسمى (الوسيط) ، أما إذا كان عددها زوجيا فإن الوسيط هو نصف مجموع القيمتين الوسطيين .
اكتب برنامجا لحساب الوسيط لكلتا الحالتين .

8. اكتب برنامجا يقوم بقراءة مصفوفتين ذواتا N صف M عمود ويطبع مجموعهما.

9. اكتب برنامجا يقرأ مصفوفة a ذات N صف و M عمود، والمصفوفة b ذات M صف و L عمود، ويقوم بحساب المصفوفة c حيث

$$c = a*b$$

وذلك حسب التعريف التالي:

$$c_{ij} = \sum a_{ik} \quad b_{kj}$$

حيث $\sum$ ترمز لعملية الجمع من k=1 إلى k=M ، وعلما بـــأن الـــمـــصفوفة الناتجة من حاصل الضرب تتكون من N صف و L عمود .

10 . اكتب برنامجا يقوم بقراءة نضيد string وحرف c وعدد صـــحيح m ، حيث m أقل من طول النضيد ، ويقوم البرنامج بإدراج insert الحرف c فـــي الموقع m من النضيد . مثال : النضيد " univrsity " والحرف e يدرج فـــي الموقع 4 من النضيد فيصبح "university " .

11. أعد البرنامج في تمرين (10) ولكن لغرض إلغاء حرف بدلاً من إضـــافة حرف.